Introducción

El objetivo del programa es desencriptar y descomprimir archivos aplicando programación inversa para recuperar el texto original. El sistema revierte transformaciones codificadas en archivos auxiliares.txt, empleando XOR con clave de 1 byte y de rotación de bits para la desencriptación, y detecta entre dos métodos de compresión (RLE y LZ78) para descomprimir el texto resultante.

a)Análisis del problema y consideraciones para la alternativa de solución propuesta

La desencriptación de archivos.txt y la descompresión

1. Análisis inicial

   para el análisis inicial se tenía en cuenta el orden presentado en el texto guía para poder desarrollar el programa, iniciando con un desencriptación iterando cada byte en el archivo a una rotación a la derecha, para cada byte del mensaje encriptado enc[i] se aplica:

   a. XOR con una llave de 1 byte key
   b. Rotación a la derecha rot bytes

   Si se dispone de una pista, se prueban combinaciones la key de 0 a 255 y solo 8 valores por el tamaño del byte , llave XOR de 1 byte, entonces:

   > Se desencripta temporalmente todo el buffer, luego se prueba si el formato es correcto para una descompresión en RLE y se busca la primera ocurrencia exacta de la pista, alguna de estas dos condiciones no se cumple, se prueba si el formata para LZ78 es correcto mientras se va descomprimiendo para esta, y se vuelve a buscar la pista en el resultado. El criterio de parada éxito se detiene en cuanto se encuentre la pista, ya que se han determinado rot y key, si el criterio de parada es fallo si se agotan las 8×256 combinaciones sin encontrar la pista se lanza un mensaje de error y no se guarda ninguna información.

   Se consideró otra solución que fue comprimir la pista y buscarla en cada desencriptación, pero de descarto esta posibilidad porque al efectuar este proceso no se logra adquirir una correspondencia exacta al arreglo que corresponde al original comprimido:

   En RLE no tenemos seguridad si la primera letra es parte de una repetición más grande o si es la primera y de forma similar en la última:

   > Ejemplo:
   > si la parte donde se tomo la pista fue : sssaaaawwwddd y el pedazo original era awdasssssssaaaawwwddddddwd, la pista comprimida seria 3s4a3w3d pero en el archivo original comprimido sería la siguiente : 1a1w1d1a6s4a3w6d1w1d, como se ve, no habría una correspondencia completa, por lo tanto se pierde seguridad.

En el caso de LZ78, tampoco es conveniente comprimir la pista y luego buscarla en el archivo comprimido. El motivo es que el algoritmo genera referencias a diccionario que dependen del

contexto previo, por lo que la misma secuencia puede codificarse de manera diferente según el momento en que aparezca.

Ejemplo:
Supongamos que la pista a buscar en texto plano es: abcabc

Si la pista aparece al inicio de un texto, LZ78 podría comprimirla como:

(0,a) (0,b) (0,c) (1,c) (3,c)

Pero si la misma secuencia aparece más adelante, el diccionario ya contiene otras entradas y la compresión podría dar como resultado algo distinto, por ejemplo:

(0,a) (0,b) (0,c) (4,3)

Aunque ambas secuencias se descomprimen en el mismo texto "abcabc", en el archivo comprimido los códigos no son iguales.

Esto significa que, al comprimir la pista y buscarla directamente en el flujo comprimido, no existe garantía de encontrar una correspondencia exacta, ya que el resultado depende del estado del diccionario en el momento de la compresión. Por esta razón, se descarta esta opción, al igual que en el caso de RLE.

Esquema de tareas en el desarrollo de los algoritmos

Análisis del problema

- Comprensión del flujo general del programa.
- Identificación de los algoritmos de compresión a implementar (RLE y LZ78).
- Revisión de fundamentos teóricos (documentación, video LZ78, ejemplo RLE).

Diseño de la solución

- Definición de estructuras de datos para entradas/salidas.
- Definición de funciones específicas para cada algoritmo.

Implementación

- Creación de módulos iniciales .h y .cpp.
- Codificación del algoritmo RLE (compresión y descompresión).
- Codificación del algoritmo LZ78 (compresión y descompresión).
- Pruebas con casos de prueba y ajustes.

Pruebas y validación

- Desencriptación de textos de prueba.
- Verificación de errores y correcciones.

Análisis de resultados

- Evaluación de la efectividad de cada algoritmo.

Integración final

- Fusión con funciones de desencriptación.
- Corrección de formatos.
- Implementación de funciones de almacenamiento.
- Pruebas finales.

Algoritmos implementados

1. **Algoritmo de descompresión:**
    - En LZ78 se usa un algoritmo que primero recrea el diccionario original con las cadenas del texto. Después suma la longitud de cada cadena para calcular la longitud del texto original. Con la longitud del texto original y el diccionario se vuelve a recorrer el texto comprimido y se rearma el texto original
    - Para el RLE se recorre el arreglo sumando las cantidades de los indicadores de repeticiones para calcular la longitud del texto original. Con esta información se lee otra vez el texto comprimido para repetir la cantidad de veces indicada el carácter correspondiente.
2. **Algoritmo de desencriptación y descompresión de la clave XOR, ROT y busqueda del metodo de compresion:**
    - Se recorren todas las posibles combinaciones para una clave hexadecimal de un byte (255) y para cada una se aplica una rotación en el rango de $0 < x < 8$ para cada posible caso se descomprime con el algoritmo de descompresión RLE primero y se verifica el formato al mismo tiempo, si el formato es correcto se utiliza un algoritmo de búsqueda en lineal que cuando encuentra una coincidencia total con la pista termina el ciclo. Si no funciona el formato para RLE, se prueba con el algoritmo de descompresión LZ78 y se hace la búsqueda de la pista.

d) Problemas de desarrollo que se afrontaron

Durante el proyecto se presentaron distintas dificultades:

1. Diseño de la lógica general

   Definir el flujo paso a paso correcto fue importante, las ideas iniciales sobre el orden de

operaciones (desencriptar/descomprimir) no produjeron la salida esperada y se reestructuró el codigo.

2. Manejo de bytes y formatos
   En RLE se esperaba un formato “byte, byte, char”, pero se implementó inicialmente una función que leía “byte, char”, lo que impedía la descompresión (salidas vacías o corruptas). Se corrigió el verificar impresión con el formato de LZ78 que si se esperaba que fuera “byte, byte, char” para respetar la especificación de tres campos y se añadió validación estricta.
3. Manejo de tipos de datos
   Al principio se hacían conversiones entre char y unsigned char, lo cual con el avance genero conflictos, especialmente a la hora de leer archivos
4. Gestión de memoria
   Se detectaron fugas en buffers temporales durante la desencriptación (especialmente en pruebas de key, rotación, lo que degrada el rendimiento.
5. Validación y robustez
   Diferencias de interpretación entre char y unsigned char afectaron la manera en la que se comunicaban las diferentes funciones y se leen los archivos

e) Evolución de la solución, resultados y consideraciones para la implementación

Evolución

El desarrollo del proyecto se llevó a cabo de manera progresiva y estructurada, siguiendo un flujo de trabajo que permitió construir el programa final de forma modular y funcional.

En una primera etapa se definieron dos ramas de trabajo principales: desencriptación y descompresión. Estas ramas se desarrollaron de manera paralela en módulos separados: el módulo funciones, enfocado en la lógica de desencriptación, y el módulo utilidades, orientado a la compresión y descompresión de datos.

Posteriormente, se implementó el módulo files, encargado de la lectura y escritura de archivos en distintos formatos, lo que facilitó la integración de los datos procesados por ambas ramas.

Una vez consolidadas las bases de cada componente, se procedió a la unión de las ramas de trabajo, lo que implicó reorganizar el código en módulos más específicos según su función:

Compresión / Descompresión - Desencriptación - Búsqueda de patrones - Manejo de archivos

Esta reorganización jerárquica permitió aislar errores, mejorar la depuración y garantizar que cada módulo trabajara de forma independiente pero coordinada dentro del sistema.

Durante esta integración se solucionaron incompatibilidades y conflictos entre funciones, se revisó la robustez de la estructura general del programa y se optimizó el manejo de memoria en cada módulo. De esta manera, el programa alcanzó una versión estable y funcional, capaz de ejecutar todo el proceso completo desde la lectura de archivos hasta la obtención de resultados finales.

Resultados:

El desarrollo del proyecto permitió integrar correctamente los módulos de desencriptación y descompresión en un único programa funcional. Se logró la lectura y escritura adecuada de archivos, la correcta separación de responsabilidades por módulos y la validación de que el sistema procesa los datos según lo esperado. Finalmente, el programa mostró ser robusto y eficiente en el manejo de memoria y en la organización de las funciones.

Consideraciones para la implementación

- Mantener la separación de funciones por módulos para facilitar la depuración.
- Documentar las rutas y nombres de archivos, ya que el sistema depende de un orden específico.
- Validar las entradas antes de ejecutar los algoritmos para evitar errores en la ejecución.
- Revisar el uso de memoria en cada módulo para garantizar eficiencia.
- En la eficiencia migrar componentes críticos a estructuras más eficientes (p. ej., tablas precalculadas de rotación) y/o permitir librerías de contenedores cuando se autorice.
- Pruebas: incluir pruebas automatizadas por archivo, con verificación byte a byte.
- Robustez: manejar casos límite como lo son archivos truncados, conteos RLE inválidos, referencias LZ78 fuera de rango con mensajes de error claros.
- En la memoria mantener reutilización de buffers y liberar en un único punto.